Team 13

SCALA

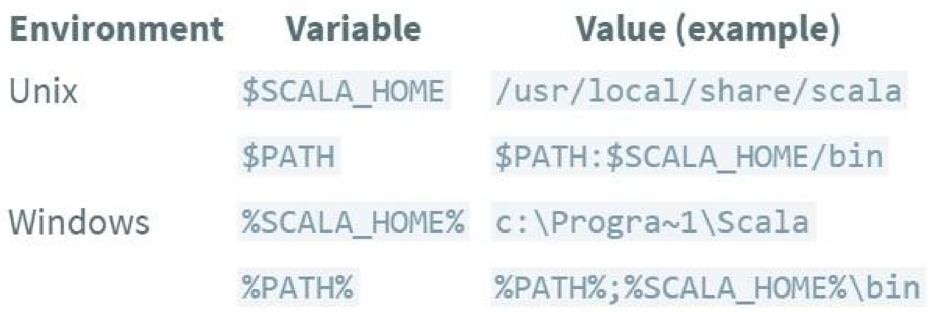
Christopher Mar
Mohit Kumar
Rebecca Goveia

History of Scala

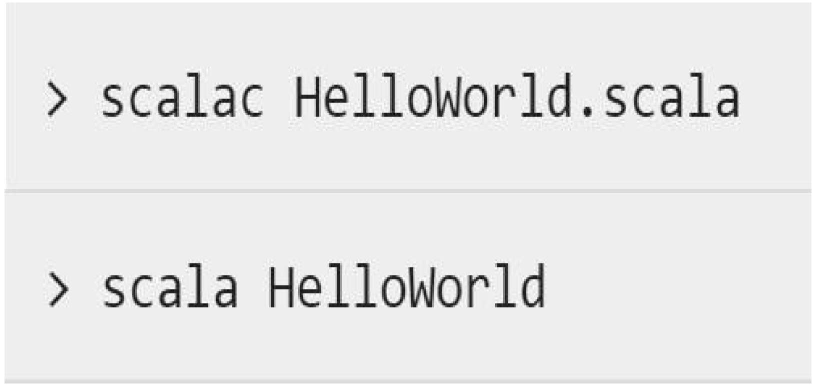
- Scala means **Sca**lable **La**nguage.
- Why Scala ?
- Java -> Funnel -> Scala
- Martin Odersky started designing Scala in 2001.
- Released in 2003.
- Interoperable with Java.

Compiler

- *scalac*
- http://scala-lang.org/download/
- Current version - 2.11.5
- https://github.com/scala/scala



Environment	Variable	Value (example)
Unix	`$SCALA_HOME`	`/usr/local/share/scala`
	`$PATH`	`$PATH:$SCALA_HOME/bin`
Windows	`%SCALA_HOME%`	`c:\Progra~1\Scala`
	`%PATH%`	`%PATH%;%SCALA_HOME%\bin`



```
> scalac HelloWorld.scala
```

```
> scala HelloWorld
```

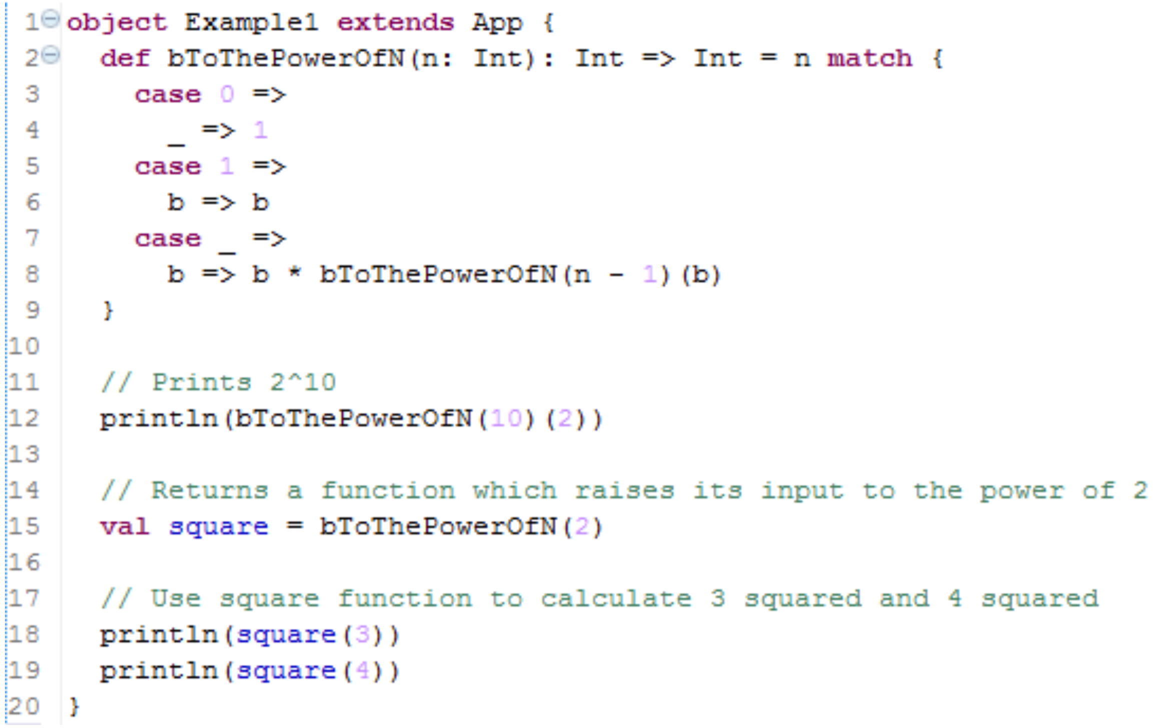
Some Features of Scala

- Object Functional Programming language
- Strong Static typing
- Supports Higher Order functions
- Immutability
- Operator overloading
- Pattern matching
- Type inference

Companies

Twitter	In 2009, Backend shifted from Ruby to Scala
Gilt	Uses Scala and Play Framework
Foursquare	Uses Scala and Lift
Coursera	Uses Scala and Play Framework
The Guardian	Java to Scala
UBS	Approved Scala for general production
LinkedIn	Uses Scalatra microframework
Verizon	Planning to make next generation framework using Scala

Example Programs



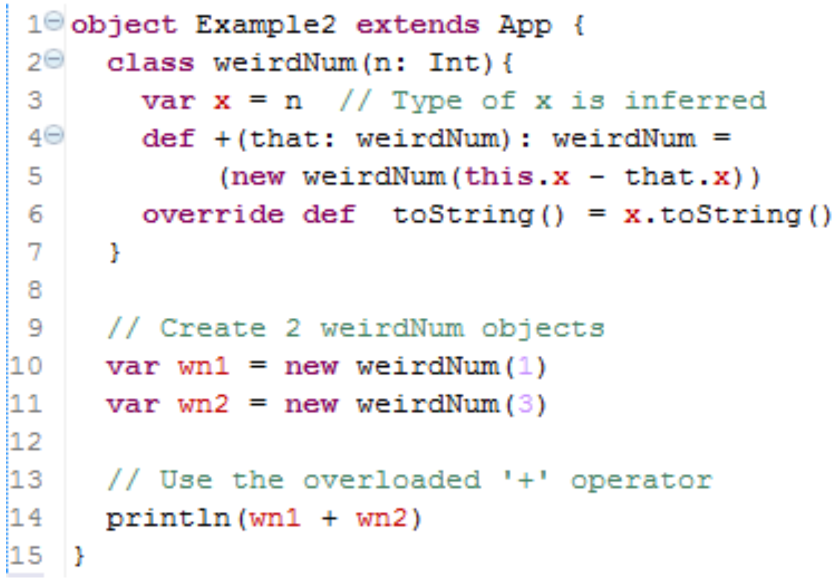
```scala
object Example1 extends App {
  def bToThePowerOfN(n: Int): Int => Int = n match {
    case 0 =>
      _ => 1
    case 1 =>
      b => b
    case _ =>
      b => b * bToThePowerOfN(n - 1)(b)
  }

  // Prints 2^10
  println(bToThePowerOfN(10)(2))

  // Returns a function which raises its input to the power of 2
  val square = bToThePowerOfN(2)

  // Use square function to calculate 3 squared and 4 squared
  println(square(3))
  println(square(4))
}
```

Example Programs



```
object Example2 extends App {
  class weirdNum(n: Int){
    var x = n  // Type of x is inferred
    def +(that: weirdNum): weirdNum =
        (new weirdNum(this.x - that.x))
    override def  toString() = x.toString()
  }

  // Create 2 weirdNum objects
  var wn1 = new weirdNum(1)
  var wn2 = new weirdNum(3)

  // Use the overloaded '+' operator
  println(wn1 + wn2)
}
```

Comparison

Since Scala borrows heavily from Java, lets compare the two.

JAVA	SCALA
In Java, **every value** is an **object**, except for primitives. Eg.: int, char, boolean etc.	In Scala, **all values** are **objects**, which the compiler turns into primitives to improve efficiency.

JAVA	SCALA

Java is **statically typed**, i.e. variables can hold values of it's type only.
 	
Example:
`int x = 13;`
`x = "hello world" // error`
	Scala does not require that you declare a type but it is statically typed and makes use of **Type Inference** to determine errors.
	
	For example:
	`var x = 13;`
	`x = "hello world"; // error`

JAVA

Java is **verbose**.

For example: class Book

```
public class Book {
    private String title;
    private String author;
    public Book (String title,
String author) {
        this.title = title;
        this.author = author;
    }
// Create Getters and Setters
}
```

SCALA

Scala cuts down on verbosity.

The same Book class can be represented as follows:

```
class Book(var title:
String, var author: String)
```

JAVA

In Java, a method which returns an object may return *`null`*.

For example:

```
import java.util.HashMap;

HashMap<String, String>
nicknames = new HashMap();
nicknames.put(“Rebecca”,
“Becky”);
nicknames.put(“Rachel”,
null);
```

SCALA

In Scala, if a method *could* return “nothing,” make it return an `Option` object, which is either `Some(`*`theObject`*`)` or `None`

```
import scala.collection.mutable.
HashMap

val nicknames = new HashMap[String,
String]
nicknames.put(“Rebecca”, “Becky” )
nicknames.put(“Rachel”, null)
```

Now to retrieve values:

```
nicknames.get(“Rebecca”);  -->
Becky
nicknames.get(“Rachel”); --> null
```

Now if the key does not exist,

```
nicknames.get(“Rhea”);  --> null
```

So if the key or value doesn’t exist, both cases return null. We can use Java’s built-in `containsKey()` to check if the key exists. All this adds to code verbosity.

Now to retrieve values:

```
scala> nicknames.get( "Rebecca" )
res1: Option[String] = Some(Becky)

scala> nicknames.get( "Rachel" )
res2: Option[String] = Some(null)
```

Now if the key does not exist,

```
scala> nicknames.get( “Rhea" )
res3: Option[String] = None
```

In this case, we get back a None type, meaning that the key doesn't exist at all.

JAVA	SCALA

Java has methods and operators (+, -, >..etc.), both behave differently and have different syntax.
 	
Java does **not** support **Operator Overloading**. The only exception may be the '+' operator used for String concatenation.
	In Scala, an operator is actually a method. The difference becomes evident based on how you use them.
	
	`val = 7 + 6 // Scala calls + defined in Int`
	`val = (7).+(6) // + is used as a method and not an operator as in the first case.`

Other languages like Groovy and Clojure also use the JVM, so lets look at their comparison.

SCALA	**GROOVY**	**CLOJURE**
Statically typed	Dynamically typed	Dynamically typed
Combines both paradigms of object-oriented and functional programming	Strongly object-oriented, focused on reducing verbosity	Object-oriented programming is deemphasized while functional programming is the main focus.
Inherits syntax from Java	Inherits syntax from Java	Inherits syntax from Lisp

References

1. http://www.scala-lang.org/what-is-scala.html
2. http://www.artima.com/weblogs/viewpost.jsp?thread=163733
3. http://docs.scala-lang.org/tutorials/
4. http://en.wikipedia.org/wiki/Scala_%28programming_language%29
5. http://www.cis.upenn.edu/~matuszek/cis700-2010/Lectures/01-scala-intro.ppt
6. http://www.scala-lang.org/docu/files/ScalaTutorial.pdf
7. http://www.toptal.com/scala/why-should-i-learn-scala